# Chapter 1

# Machine Learning Methods for Automatic Image Colorization

GUILLAUME CHARPIAT

Pulsar Project
INRIA
Sophia-Antipolis, France
Email: Guillaume.Charpiat@sophia.inria.fr

ILJA BEZRUKOV
YASEMIN ALTUN
MATTHIAS HOFMANN
BERNHARD SCHÖLKOPF

Max Planck Institute for Biological Cybernetics
Tübingen, Germany
Email: Firstname.Name@tuebingen.mpg.de

---

## 1.1 Introduction

Automatic image colorization consists in adding colors to a new greyscale image without any user intervention. The problem, stated like this, is ill-posed, in the sense that one cannot guess the colors to assign to a greyscale image without any *prior* knowledge. Indeed, many objects can have different colors: not

only artificial, plastic objects can have random colors, but natural objects like tree leaves can have various nuances of green and turn brown during autumn, without a significant change of shape.

The color *prior* most often considered in the literature is the user: many approaches consist in letting the user determine the color of some areas and then in extending this information to the whole image, either by pre-computing a segmentation of the image into (hopefully) homogeneous color regions, or by spreading color flows from the user-defined color points. This last method supposes a definition of the difficulty of the color flow to go through each pixel, usually estimated as a simple function of local greyscale intensity variations, as in Levin *et al*. [1], in Yatziv and Sapiro [2], or in Horiuchi [3], or by predefined thresholds to detect color edges [4]. However, we have noticed, for instance, that the simple, efficient framework by Levin *et al*. cannot deal with texture examples such as in figure 1.1, whereas simple oriented texture features such as Gabor filters would have obviously solved the problem[1]. This is the reason why we need to consider texture descriptors. More generally, each hand-made criterion for edge estimation has its drawbacks, and therefore we will *learn* the probability of color change at each pixel instead of setting a hand-made criterion.

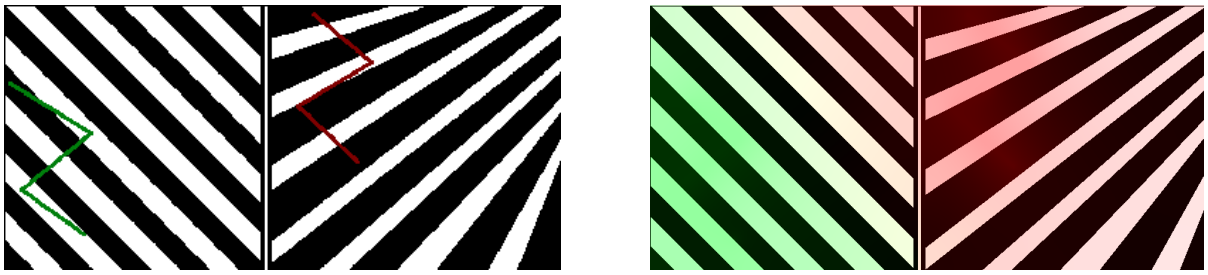

Figure 1.1: Failure of standard colorization algorithms in presence of texture. Left: manual initialization; right: result of Levin *et al*.'s approach. In spite of the general efficiency of their simple method (based on the mean and standard deviation of local intensity neighborhoods), texture remains difficult to deal with. Hence the need of texture descriptors and of learning edges from color examples.

User-based approaches present the advantage that the user can interact, add more color points if needed, until a satisfying result is reached, or even place color points strategically in order to give indirect information on the location of color boundaries. Our method can easily be adapted to incorporate such user-provided color information. The task of providing first a fully automatic colorization of the image, before a possible user intervention if necessary, is however much harder.

Some recent attempts of *predicting* the colors gave mixed results. For instance the problem has been studied by Welsh *et al*. [5]; it is also one of the applications presented by Hertzmann *et al*. in [6]. However the framework developed in both cases is not mathematically expressed, in particular it is not clear whether an energy is minimized, and the results shown seem to deal with only a few colors, with many small artifacts

[1]Their code is available at `http://www.cs.huji.ac.il/~weiss/Colorization/`.

probably due to the lack of a suitable spatial coherency criterion. An other notable study is by Irony *et al*. [7]: it consists in finding a few points in the image where a color prediction algorithm reaches the highest confidence, and then in applying Levin's approach as if these points were given by the user. Their approach of color prediction is based on a training set of colored images, partially segmented by the user into regions. The new image to be colored is then automatically segmented into locally homogeneous regions whose texture is similar to one of the colored regions previously observed, and the colors are transferred. Their method reduces the effort required of the user but still requires a manual pre-processing step. To avoid this, they proposed to segment the training images automatically into regions of homogeneous texture, but fully automatic segmentation (based on texture or not) is known to be a very hard problem. In our approach we will not rely on an automatic segmentation of the training or test images but we will build on a more robust ground.

Irony's method also brings more spatial coherency than previous approaches, but the way coherency is achieved is still very local since it can be seen as a one-pixel-radius filter, and furthermore it relies partly on an automatic segmentation of the new image.

But above all, the latter method, as well as all other former methods in the history of colorization (to the knowledge of the authors), cannot deal with the case where an ambiguity concerning the colors to assign can be resolved only at the global level. The point is that local predictions based on texture are most often very noisy and not reliable, so that information needs to be integrated over large regions to become significant. Similarly, the performance of an algorithm based on texture classification, such as Irony's one, would drop dramatically with the number of possible texture classes, so that there is a real need for robustness against texture misclassification or noise. In contrast to previous approaches, we will avoid to rely on very-local texture-based classification or segmentation and we will focus on more global approaches.

The color assignment ambiguity also happens when the shape of objects is relevant to determine the color of the whole object. More generally, it appears that boundaries of objects contain much not-immediately-usable information, such as the presence of edges in the color space, and also contain significant details which can help the identification of the whole object, so that the colorization problem cannot be solved at the local level of pixels. In this chapter we try to make use of all the information available, without neglecting any low probability at the local level which could make sense at the global level.

Another source for prior information is motion and time coherency in the case of video sequences to be colored [1]. Though our framework can easily be extended to the film case and also benefit from this information, we will deal only with still images in this chapter.

First we present, in section 1.2, the model chosen for the color space, as well as the model for local greyscale texture. Then, in section 1.3, we state the problem of *automatic image colorization* in machine learning terms, explain why the naive approach, which consists in trying to predict directly color from

texture, performs poorly, and we show how to solve the problem by learning multimodal probability distributions of colors, which allows the consideration of different possible colorizations at the local level. We also show how to take into account user-provided color landmarks when necessary. We start section 1.4 with a way to *learn* how likely color variations are at each pixel of the image to color. This defines a spatial coherency criterion which we use to express the whole problem mathematically. We solve a discrete version of it via graph cuts, whose result is already very good, and refine it to solve the original continuous problem. Finally, in section 1.5, we propose a discriminative way to model spatial coherency, with structured prediction learning.

## 1.2 Model for colors and greyscale texture

We first model the basic quantities to be considered in the image colorization problem.

Let $I$ denote a greyscale image to be colored, $\mathbf{p}$ the location of one particular pixel, and $C$ a proposition of colorization, that is to say an image of same size as $I$ but whose pixel values $C(\mathbf{p})$ are in the standard RGB color space. Since the greyscale information is already given by $I(\mathbf{p})$, we should add restrictions on $C(\mathbf{p})$ so that computing the greyscale intensity of $C(\mathbf{p})$ should give $I(\mathbf{p})$ back. Thus the dimension of the color space to be explored is intrinsically 2 rather than 3.

We present in this section the model chosen for the color space, our way to discretize it for further purposes, and how to express continuous probability distributions of colors out of such a discretization. We also present the feature space used for the description of greyscale patches.

### 1.2.1 $L$-$a$-$b$ color space

In order to quantify how similar or how different two colors are, we need a metric in the space of colors. Such a metric is also required to associate to any color a corresponding grey level, *i.e*. the closest unsaturated color. This is also at the core of the color coherency problem: an object with a uniform reflectance will show different colors in its illuminated and shadowed parts since they have different grey levels, so that we need a way to define robustly colors against changes of lightness, that is to consider how colors are expected to vary as a function of the grey level, *i.e*. how to project a dark color onto the subset of all colors who share a particular brighter grey level.

The psychophysical $L$-$a$-$b$ color space was historically designed so that the Euclidean distance between the coordinates of any colors in this space approximates as well as possible the human perception of distances between colors. The transformation from standard RGB colors to L-a-b consists in first applying gamma correction, followed by a linear function in order to obtain the $XYZ$ color space, and then by another highly non-linear application which is basically a linear combination of the cubic roots of the coordi-

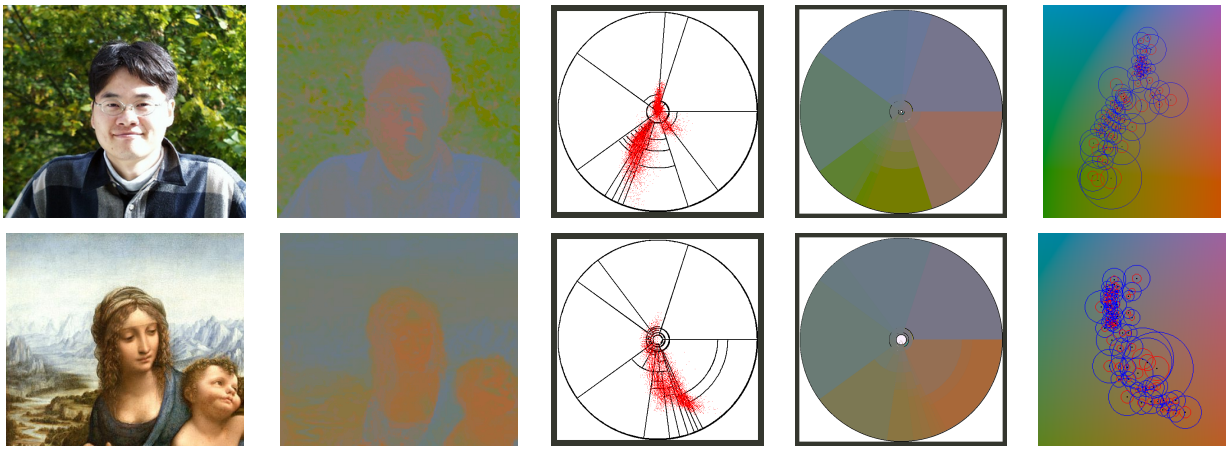

Figure 1.2: Examples of color spectra and associated discretizations. For each line, from left to right: color image; corresponding 2D colors; location of the observed 2D colors in the $ab$-plane (a red dot for each pixel) and the computed discretization in color bins; color bins filled with their average color; continuous extrapolation: influence zones of each color bin in the $ab$-plane (each bin is replaced by a Gaussian, whose center is a black dot; red circles indicate the standard deviation of colors within the color bin, blue ones are three times bigger).

nates in $XYZ$. We refer to Lindbloom's website[2] for more details on color spaces or for the exact formulas. The $L$-$a$-$b$ space has three coordinates: $L$ expresses the luminance, or lightness, and is consequently the greyscale axis, whereas $a$ and $b$ stand for two orthogonal color axes.

We choose the $L$-$a$-$b$ space to represent colors since its underlying metric has been designed to express color coherency. In the following, by *grey level* and *2D color* we will mean respectively $L$ and $(a, b)$.

With the previous notations, since we already know the grey level $I(\mathbf{p})$ of the color $C(\mathbf{p})$ to predict at pixel $\mathbf{p}$, we search only for the remaining 2D color, denoted by $ab(\mathbf{p})$.

### 1.2.2 Discretization of the color space

This subsection contains more technical details and can be skipped in the first reading.

In section 1.3 we will temporarily need a discretization of the 2D color space. Instead of setting a regular grid, we define a discretization adapted to the color images given as examples so that each color bin will contain approximately the same number of observed pixels with this color. Indeed, some entire zones of the color space are useless and we prefer to allocate more color bins where the density of observed colors is higher, so that we can have more nuances where it makes statistical sense. Figure 1.2 shows the densities of colors corresponding to some images, as well as the discretizations in 73 bins resulting from these densities. We colored each color bin by the average color of the points in the bin. To obtain these discretizations, we

[2] `http://brucelindbloom.com/`

used a polar coordinate system in $ab$ and cut recursively color bins with highest numbers of points at their average color into 4 parts. Discussing the precise discretization algorithm is not relevant here provided it makes sense statistically; we could have used $K$-means for instance.

Further, in section 1.4, we will need to express densities of points over the whole plane $ab$, based on the densities computed in each color bin. In order to interpolate continuously the information given by each color bin $i$, we place Gaussian functions on the average color $\mu_i$ of each bin, with a standard deviation proportional to the statistically observed standard deviation $\sigma_i$ of points of each color bin (see last column of figure 1.2). Then we interpolate the original density $d(i)$ to any point $\mathbf{x}$ in the $ab$ plane by:

$$d_G(\mathbf{x}) = \sum_i \frac{1}{\pi(\alpha\sigma_i)^2} \, e^{-\frac{(\mathbf{x}-\mu_i)^2}{2(\alpha\sigma_i)^2}} \, d(i)$$

We found experimentally that considering a factor $\alpha \approx 2$ improved significantly the distribution aspect. Cross-validation could be used to determine the optimal $\alpha$ for a given training set.

### 1.2.3 Greyscale patches and features

The grey level of one pixel is clearly not informative enough to decide which 2D color we should assign to it. The hope is that texture and local context will give more clues for such a task (see section 1.3). In order to extract as much information as possible to describe local neighborhoods of pixels in the greyscale image, we compute SURF descriptors [8] at three different scales for each pixel. This leads to a vector of 192 features per pixel. In order to reduce the number of features and to condense the relevant information, we apply Principal Component Analysis (PCA) and keep the first 27 eigenvectors, to which we add, as supplemental components, the pixel grey level as well as two biologically inspired features: a weighted standard deviation of the intensity in a $5 \times 5$ neighborhood (whose meaning is close to the norm of the gradient), and a smooth version of its Laplacian. We will refer to this 30-dimensioned vector, computed at each pixel, as *local description* in the following, and denote it by $\mathbf{v}$ or $\mathbf{w}$.

## 1.3 Color prediction

Now that we have modelled colors as well as local descriptions of greyscale images, we can start stating the image colorization problem. Given a set of examples of color images, and a new greyscale image $I$ to be colored, we would like to extract knowledge from the training set to predict colors $C(\mathbf{p})$ for the new image.

### 1.3.1 Need for multimodality

One could state this problem in simple machine learning terms: learn the function which associates to any local description of greyscale patches the right color to assign to the center pixel of the patch. This could be

achieved by kernel regression tools such as Support Vector Regression (SVR) or Gaussian Processes [9].

There is an intuitive reason why this would perform poorly. Many objects can have different colors, for instance balloons at a fair could be green, red, blue, *etc*., so that even if the task of recognizing a balloon was easy and that we knew that we should use colors from balloons to color the new one, a regression would recommend the use of the average value of the observed balloons, *i.e*. grey. The problem is however not specific to objects of the same class. Local descriptions of greyscale patches of skin or sky are very similar, so that learning from images including both would recommend to color skin and sky with purple, without considering the fact that this average value is never probable.

We therefore need a way to deal with multi-modality, *i.e*. to predict different colors if needed, or more exactly, to predict at each pixel the *probability* of *every* possible color. This is in fact the conditional probability of colors *knowing* the local description of the greyscale patch around the pixel considered. We will also be interested in the confidence in these predictions in order to know whether some predictions are more or less reliable than others.

### 1.3.2 Probability distributions as density estimations

The conditional probability of the color $c_i$ at pixel $\mathbf{p}$ knowing the local description $\mathbf{v}$ of its greyscale neighborhood can be expressed as the fraction, amongst colored examples $e_j = (\mathbf{w}_j, c(j))$ whose local description $\mathbf{w}_j$ is similar to $\mathbf{v}$, of those whose observed color $c(j)$ is in the same color bin $B_i$. We thus have to estimate densities of points in the feature space of grey patches. This can be accomplished with a Gaussian Parzen window estimator:

$$p(c_i|\mathbf{v}) = \Big( \sum_{\{j\,:\,c(j)\in B_i\}} k(\mathbf{w}_j, \mathbf{v}) \Big) \Big/ \sum_j k(\mathbf{w}_j, \mathbf{v})$$

where $k(\mathbf{w}_j, \mathbf{v}) = e^{-(\mathbf{w}_j - \mathbf{v})^2/2\sigma^2}$ is the Gaussian kernel. The best value for the standard deviation $\sigma$ can be estimated by cross-validation on the densities. With this framework we can express how reliable the probability estimation is: its confidence depends directly on the density of examples around $\mathbf{v}$, since an estimation far from the clouds of observed points loses signification. Thus, the confidence in a probability prediction is the density in the feature space itself:

$$p(\mathbf{v}) \propto \sum_j k(\mathbf{w}_j, \mathbf{v})$$

In practice, for each pixel $\mathbf{p}$, we compute the local description $\mathbf{v}(\mathbf{p})$, but we do not need to compute the similarities $k(\mathbf{v}, \mathbf{w}_j)$ to all examples in the training set: in order to save computational time, we only search for the $K$-nearest neighbors of $\mathbf{v}$ in the training set, with $K$ sufficiently large (as a function of the $\sigma$ chosen), and estimate the Parzen densities based on these $K$ points. In practice we choose $K = 500$, and thanks to

fast nearest neighbor search techniques such as kD-tree[3], the time needed to compute all predictions for all pixels of a $50 \times 50$ image is only 10 seconds (for a training set of hundreds of thousands of patches) and this scales linearly with the number of test pixels. Note that we could also have used sparse kernel techniques such as SVR to estimate, for each color bin, a regression between the local descriptions and the probability of falling into the color bin. We refer more generally to [9] for details and discussions about kernel methods.

### 1.3.3 User-provided color landmarks

We can easily consider user-provided information such as the color $c$ at pixel $\mathbf{p}$ in order to modify a colorization obtained automatically. We set $p(c|\mathbf{p}) = 1$ and set the confidence $p(\mathbf{p})$ to a very large value. Consequently our optimization framework is still usable for further interactive colorization. A re-colorization with new user-provided color landmarks does not require the re-estimation of color probabilities, and therefore lasts only a fraction of second (see next part).

## 1.4 Learning color variations

For each pixel of a new greyscale image, we are now able to estimate the probability distribution of all possible colors (within a big finite set of colors since we discretized the color space into bins). The interest in such computation is that, if we add a spatial coherency criterion, a pixel will be influenced by its neighbors, and the choice of the best color to assign will be done accordingly to the probability distributions in the neighborhood. Since all pixels are linked by neighborhoods, even if not directly, they all interact with each other, so that the solution has to be computed globally. Indeed it may happen that, in some regions that are supposed to be homogeneous, a few different colors may seem to be the most probable ones at a local level, but that the winning color at the scale of the region is different, because in spite of its only second rank probability at the local level, it ensures a good probability *everywhere* in the whole region. The opposite may also happen: to flip a whole such region to a color, it may be sufficient that this color is considered as extremely probable at a few points with high confidence. The problem is consequently not trivial, and the issue is to find a global solution. In this section we first *learn* a spatial coherency criterion, then find a good solution to the whole problem with the help of graph cuts.

### 1.4.1 Local color variation prediction

Instead of picking randomly a prior for spatial coherence, based either on detection of edges, or on the Laplacian of the intensity, or on a pre-estimated complete segmentation, we *learn* directly how likely it is to observe a color variation at a pixel knowing the local description of its greyscale neighborhood, based on

[3] We use, without particular optimization, the TSTOOL package available at `http://www.physik3.gwdg.de/tstool/`.

a training set of real color images. The technique is similar to the one detailed in the previous section. For each example $\mathbf{w}_j$ of colored patch, we compute the norm $g_j$ of the gradient of the 2D color (in the $L$-$a$-$b$ space) at the center of the patch. The expected color variation $g(\mathbf{v})$ at the center of a new greyscale patch $\mathbf{v}$ is then:

$$g(\mathbf{v}) = \frac{\sum_j k(\mathbf{w}_j, \mathbf{v})\, g_j}{\sum_j k(\mathbf{w}_j, \mathbf{v})}.$$

Thus we now have priors both on the colors and on the color variations.

### 1.4.2 Global coherency via graph cuts

The graph cut, or max flow, algorithm is a minimization technique widely used in computer vision [10, 11] because of its suitability for many image processing problems, because of its guarantee to find a good local minimum, and because of its speed. In the multi-label case with $\alpha$-expansion [12], it can be applied to all energies of the form $\sum_i V_i(x_i) + \sum_{i\sim j} D_{i,j}(x_i, x_j)$ where $x_i$ are the unknown variables, with possible values in a **finite** set $\mathcal{L}$ of labels, where the $V_i$ are any functions, and where $D_{i,j}$ are any pair-wise interaction terms with the restriction that each $D_{i,j}(\cdot,\cdot)$ should be a metric on $\mathcal{L}$.

The reason why we temporarily discretized the color space in section 1.2 was to be able to use this technique. We formulate the image colorization problem as an optimization problem on the following energy:

$$\sum_{\mathbf{p}} V_{\mathbf{p}}(c(\mathbf{p})) \;\; + \;\lambda \sum_{\mathbf{p}\sim\mathbf{q}} \frac{|c(\mathbf{p}) - c(\mathbf{q})|_{Lab}}{g_{\mathbf{p},\mathbf{q}}} \tag{1.1}$$

where $V_{\mathbf{p}}(c(\mathbf{p})) \;=\; -\log\big(\,p\big(\mathbf{v}(\mathbf{p})\big)\,\big)\;\; p\big(c(\mathbf{p})|\mathbf{v}(\mathbf{p})\big)$ is the cost of choosing color $c(\mathbf{p})$ for pixel $\mathbf{p}$ (whose neighboring texture is described by $\mathbf{v}(\mathbf{p})$)) and where $g_{\mathbf{p},\mathbf{q}} = 2\left(g(\mathbf{v}(\mathbf{p}))^{-1} + g(\mathbf{v}(\mathbf{q}))^{-1}\right)^{-1}$ is the harmonic mean of the estimated color variation at pixels $\mathbf{p}$ and $\mathbf{q}$. An 8-neighborhood is considered for the interaction term, and $\mathbf{p} \sim \mathbf{q}$ means that $\mathbf{p}$ and $\mathbf{q}$ are neighbors.

The term $V_{\mathbf{p}}$ penalizes colors which are not probable at the local level according to the probability distributions obtained in section 1.3, with the strength depending on the confidence in the predictions. The interaction term between pixels penalizes color variation where it is not expected, according to the variations predicted in the previous paragraph.

We use the graph cut package[4] provided by [13]. The solution for a $50 \times 50$ image and 73 possible colors is obtained by graph cuts in a fraction of second and is generally already very satisfying. The computation time scales approximately quadratically with the size of the image, which is still fast, and the algorithm performs well even on massively down-scale versions of the image, so that a good initial clue can still be

[4] available at `http://vision.middlebury.edu/MRF/code/`

given quickly for very big images too. The computational costs compete with those of the fastest colorization techniques [14] while bringing much more spatial coherency.

### 1.4.3 Refinement in the continuous color space

We can now go back to the initial problem in the continuous space of colors. We interpolate probability distributions $p(c_i|\mathbf{v}(\mathbf{p}))$ estimated at each pixel $\mathbf{p}$ for each color bin $i$, to the whole space of colors with the technique described in section 1.2, so that $V_{\mathbf{p}}(c)$ is now defined for any color $c$ and not only for color bins. The energy (1.1) can consequently be minimized in the continuous space of colors. We start from the solution obtained by graph cuts and refine it with a gradient descent. This refinement step will generally not introduce huge changes like flipping of whole regions, but will bring more nuances.

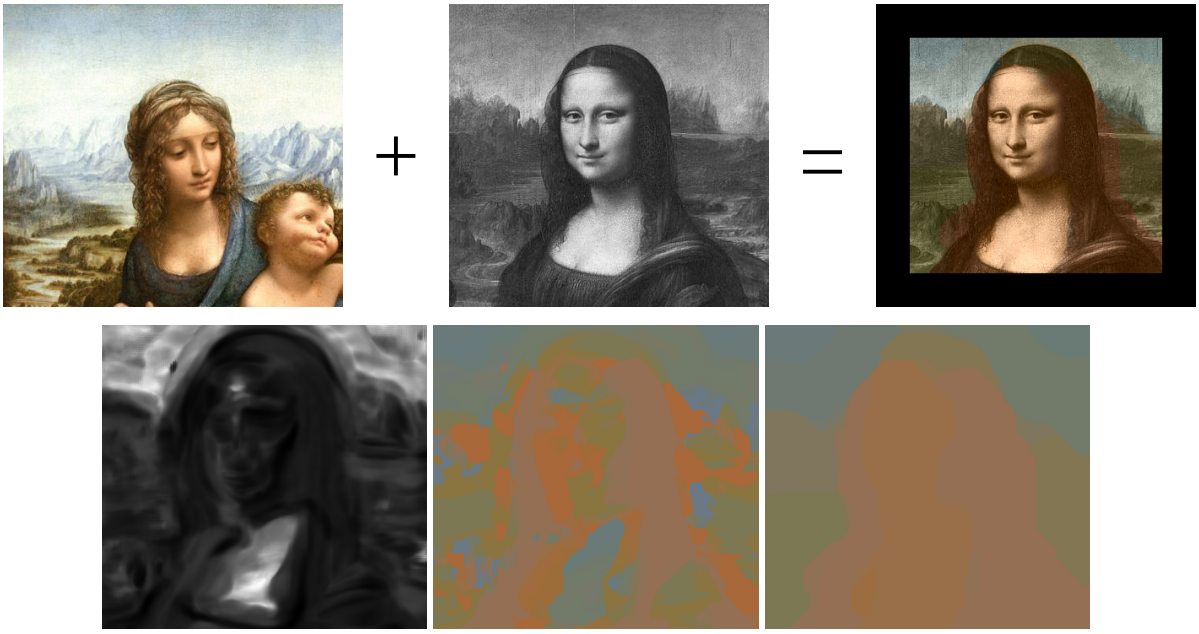

Figure 1.3: Da Vinci case: *Mona Lisa* colored with *Madonna with the yarnwinder*. The border is not colored because of the window size needed for SURF descriptors. Second line: color variation predicted (white stands for homogeneity and black for color edge); most probable color at the local level; 2D color chosen by graph cuts. Note that previous algorithms could not deal with regions such as the neck or the forehead, where blue is the most probable color at the local level because greyscale skin looks like sky. Surroundings of these regions and lower-probability colors are decisive for the final color choice.

### 1.4.4 Experiments

We now show results of automatic colorization. In figure 1.3 we colored a famous painting by Leonardo Da Vinci with another painting of his. The paintings are significantly different and textures are relatively

dissimilar. The prediction of color variation performs well and helps much to determine the boundaries of homogeneous color regions. The multimodality framework proves extremely useful in areas such as Mona Lisa's forehead or neck where the texture of skin can be easily mistaken with the texture of sky at the local level. Without our global optimization framework, several entire skin regions would be colored in blue, disregarding the fact that skin color is a second probable possibility of colorization for these areas, which makes sense at the global level since they are surrounded by skin-colored areas, with low probability of edges. We insist on the fact that the input of previous texture-based approaches is very similar to the "most probable color" prediction (second line, middle image), whereas we consider the probabilities of all possible colors at all pixels. This means that, given a certain quality of texture descriptors, we handle much more information.

In figure 1.4 we perform similar experiments with photographs of landscapes. The effect of the refinement step can be observed in the sky, where nuances of blue vary more smoothly.

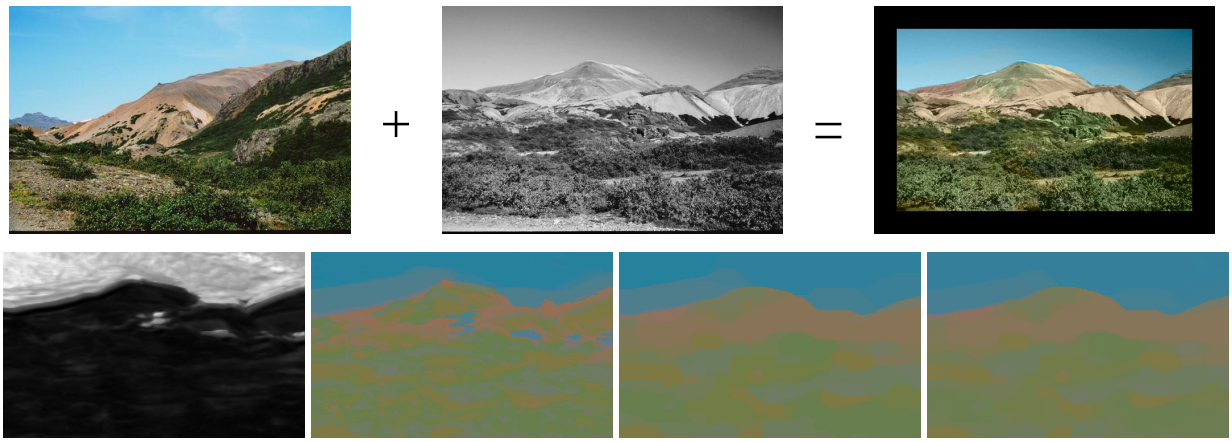

Figure 1.4: Landscape example. Same display as in figure 1.3, plus last image: colors obtained after refinement step. Note that the sky is more homogeneous, the color gradients in the sky are smoother than when obtained directly by graph cuts (previous image).

We compare our method with the one by Irony *et al.*, on their own example [7] in figure 1.5; the task is easier, and results are similar. The boundaries of our color regions fit better to the zebra contour. However, grass areas near the zebra are colored according to the grass observed at similar locations around the zebra in the training image, thus creating color halos which are visually not completely satisfactory. This bias should disappear with bigger training sets since the color of the background would become independent of zebra's presence.

In figure 1.6 we consider a very difficult task: the one of coloring an image from a Charlie Chaplin movie, with many different objects and textures, such as a brick wall, a door, a dog, a head, hands, a loose

suit... Because of the number of objects, and because of their particular arrangement, it is unlikely to find a single color image with a similar scene that we would use as a training image. Thus we consider a small set of three different images, each of which shares a partial similarity with Charlie Chaplin's image. The underlying difficulty is that each training image also contains parts which should not be re-used in this target image. The result is however promising, considering the training set. Dealing with bigger training datasets should slow the process only logarithmically, during the kD-tree search.

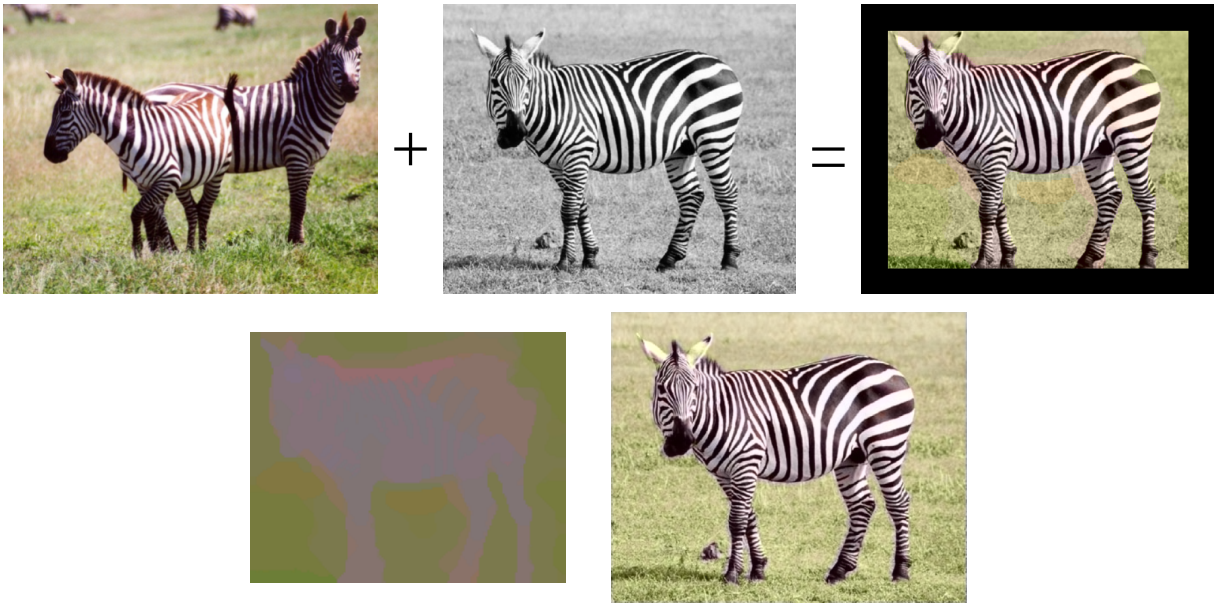

Figure 1.5: Comparable results with Irony *et al*. on their own example[7]. First line: our result. Second line, left: our predicted 2D colors; right: Irony *et al*.'s result with the assumption that this is a binary classification problem. Note that this example they chose could be solved easily without texture consideration since there is a simple correspondence between grey level intensity and color. Concerning our result, the color variations in the grass around the zebra are probably due to the influence of the grass color for similar patches in the training image, and this problem should disappear with a larger training set : we should not try to learn colors from a single overfitted, carefully-chosen training image but rather check how the colorization scales with the number of training images and with their non-perfect suitability. Note however, if you zoom in, that the contour of the zebra matches exactly boundaries of 2D color regions (purple and pink), whereas in Irony *et al*.'s result, a several pixel wide band of grass around the zebra's legs and abdomen are colored as if they were part of the zebra.

### 1.4.5 Experiments in the interactive framework

*It could be nice to add some results related to section 1.3.3 (with user-interaction), since it has been implemented... and then a comparison with Levin* et al. *would be required, and we do not always perform*

Figure 1.6: Charlie Chaplin frame colored by three different images. Second line: prediction of color variations, most probable color at the local level, and final result. This example is particularly difficult because many different textures and colors are involved and because there does not exist color images very similar to Charlie Chaplin's one. We have chosen three different images, each of which shares a partial similarity with the target image. In spite of the complexity, the prediction of color edges or of homogeneous regions remains significant. The brick wall, the door, the head and the hands are globally well colored. The large trousers are not in the training set; the mistakes in the colors of Charlie Chaplin's dog are probably due to the blue reflections on the dog in the training image and to the light brown of its head. Both would probably benefit from larger training sets.

*better. In the general case, the very simple method by Levin* et al. *performs surprisingly well (better than ours), even though it is not designed to be able to deal with texture; but it fails on high-contrast textures like zebras's.*

### 1.4.6 Scaling with training set size (or not)

*Consideration of 20 or 50 paintings of the same painter as a training set. Not clear how this would perform.*

## 1.5 Structured prediction learning

*Yasemin & Ilja's part.*

### 1.5.1 SVM for structured output prediction

Multiclass Support Vector Machines [**?**, **?**] are widely used in the machine learning community for classification tasks, however, the output space is limited to simple class labels.

A generalization for one-dimensional sequences of labels was proposed by Altun et al.[**?**]

Tsochantaridis et al.[**?**] described a method for generalizing SVM classification to arbitrary structured outputs and a polynomial time training algorithm. The idea is to learn a discriminant function $F : X \times Y \Rightarrow \mathbf{R}$ over input/output pairs. A prediction for a label $y$ can be derived by maximizing $F$ over the space of possible labels $Y$ for a given input $x$.

$$\bar{y} = \underset{y \in Y}{\operatorname{argmax}} F(x, y; w), \tag{1.2}$$

where w denotes a parameter vector. $F$ is assumed linear in a combined representation of inputs and outputs $\Phi(x, y)$. Thus, $F(x, y; w)$ can be expressed as a dot product between the feature map $\Phi(x, y)$ and the parameter vector $w$.

$$F(x, y; w) =< w, \Phi(x, y) > . \tag{1.3}$$

The form of $\Phi$ depends on the structure of the problem and we will describe our definition in section 1.5.3.

The optimization problem which has to be solved during the training phase is as follows

$$\text{SVM}_1^{\Delta_m} : \min_{w,\zeta} \frac{1}{2}||w||^2 + \frac{C}{I}\sum_{i=1}^{I}\zeta_i; \forall i, \forall y \in Y \setminus y_i : \langle w, \Phi(x_i, y_i) - \Phi(x_i, y)\rangle \geq \Delta(y_i, y) - \zeta_i; \tag{1.4}$$

, where $\Delta(y_i, y)$ denotes the loss function. The basic idea of training is to find a small set of constraints that ensures a sufficiently accurate solution. For each training sample, a working set of constraints is maintained. In each iteration of the most violated constraint is added for each training sample to its working set. The algorithm terminates when no constraint from the working sets is violated by more than a small $\epsilon$ value.

### 1.5.2 Features

### 1.5.3 Intermodal image prediction with structured SVM

The framework of SVMstruct needs to be adapted to the specific requirements of the image prediction task. Firstly, we need to define a joint representation of the input and output images. This representation is described in section **??**. Secondly, a task-specific loss function which reflects the similarity between target images has to be defined (Section 1.5.3). Finally, we need an efficient method for maximization of the dot product between the weights and the feature map (Section **??**).

#### Feature functions

The joint feature map $\Phi(x, y)$ provides a compact representation of input image characteristics, combined with their relationship to the predicted labels. The structure is depicted in figure **??**. Basically, we can distinguish between two kinds of features. The first type provides local characteristics of a predicted pixel

label. It can be formalized as follows

$$\phi_c^t(x, y) = \{[y_t = c]\psi_r(x_t)|r = 1, \ldots, R\}\,, \tag{1.5}$$

where $x$ denotes the input image, $y$ the predicted image, $t$ the position in the image and $c$ a color label. $[\ldots]$ is the indicator function. details. $\psi_r(x_t)$ represents the $r^{th}$ feature value at the position $t$. In our setting, $r$ ranges from 1 to 30. The features we use are described in section **??**.

The second type expresses the relationship between two neighboring labels $t_1$ and $t_2$ :

$$\phi_{c_1c_2}^{t_1t_2} = [y_{t_1} = c_1 \wedge y_{t_2} = c_2]. \tag{1.6}$$

The combined feature map at location $t$ is build by a union of both types of features:

$$\Phi(x, y; t) = \{\phi_c^t(x, y)|\,\forall\, c\} \cup \{\phi_{c_1c_2}^{t_1t_2}(x, y)|\,\forall\, c_1, c_2, t_1 \sim t_2\}. \tag{1.7}$$

Finally, we compute the joint feature map $\Phi(x, y)$ by summing up the feature maps at each location $t$ in the image:

$$\Phi(x, y) = \sum_{t=1}^{T} \Phi(x, y; t). \tag{1.8}$$

**Loss functions**

In the search for the most violated constraint during the training, we need to measure the correctness of the predicted labellings. We tested three loss functions which differ in the incorporation of task-specific information formulation.

The zero-one loss is the most unspecific loss function. It returns 1 if two image labellings are equal and 0 otherwise.

The Hamming loss compares the pixel labellings on local level

$$\sum_t [y_t! = \bar{y}_t], \tag{1.9}$$

where $y_t$ is the true label at location $t$ and $\bar{y}_t$ the predicted label.

The distance loss also takes into account the perceived difference between colors

$$\sum_t \frac{||y_t - \bar{y}_t||}{\max_{a,b\in C} ||a - b||}. \tag{1.10}$$

The normalization term ensures that the local color differences are between 0 and 1.

### Label inference

A crucial part of the framework is the maximization of the dot product between the feature map and the weight vector over the space of possible labels. During training, we have to search for the most violated constraint insert and cite equation in every iteration and for every training sample, so speed is an important factor.

### Local maximization

As a baseline, we used a joint feature map without neighborhood dependencies

$$\Phi_L(x,y;t) = \{\phi_c^t(x,y) | \, \forall \, c\} Phi_L(x,y) = \sum_{t=1}^{T} \Phi(x,y;t). \tag{1.11}$$

For this feature map, the maximization can be performed locally for every pixel $t$

$$\begin{aligned} \underset{\bar{y}}{\operatorname{argmax}} \langle \Phi_L(x,\bar{y}), w \rangle &= \underset{\bar{y}}{\operatorname{argmax}} \langle \sum_{t=1}^{T} \Phi_L(x,y;t), w \rangle & (1.12) \\ &= \rangle \sum_{t=1}^{T} \underset{c}{\operatorname{argmax}} \, \Phi_L(x,y;t), w \rangle. & (1.13) \end{aligned}$$

### Global maximization

For the case of a full feature map with dependencies between neighboring labels, the maximization has to be performed globally over the whole image. We used the code from Szeliski et al. [**?**], which contains several algorithms for energy minimization over Markov Random Fields. In our studies, we used Loopy Belief Propagation (LBP) and Tree-Reweighted Message Passing (TRW-S). *I*t might make sense to try Graph Cuts, with all the improvements done on the MR-CT code, it should work. The energy function is derived from the dot product of the joint feature map and the weight vector and consists of a data energy $E_d$ for each pixel $t$ and a label $c$ and a transition/smoothness energy $E_s$ between two labels $c_1$ and $c_2$ , smoothness energy is position independent, best to change the $\psi$ definition to reflect this.

$$\begin{aligned} E_d(t,c) &= \langle w_{[cR,(c+1)R]}, \phi_c^t(x,y) \rangle & (1.14) \\ E_s(c_1,c_2) &= -s \cdot w_{c_1,c_2} \cdot f, & (1.15) \end{aligned}$$

where $s$ is a parameter to adjust the balance between both energy types and $f$ is parameter which can be used to incorporate prior knowledge about the transition costs. It can take the following values:

- 1, when no prior knowledge is used.
- Similarity between colors $1 - \frac{||c-d||}{\max_{a,b \in C} ||a-b||} + \epsilon$.
- Normalized relative frequency of transitions between $c_1$ and $c_2$Formula

### 1.5.4 Experiments

We performed various experiments with different image sets to test the performance of our method. As a baseline, we used a setup where no neighborhood interactions in training or test were considered. Fur- ther, we performed a local training without neighborhood interactions, but used color similarity as neighborhood interaction terms. Finally, we used trained neighborhood interaction terms both in training and test.

## 1.6 Discussion

We presented a new approach for automatic image colorization, which does not require any intervention by the user, except from the choice of relatively similar color images. The problem was stated mathematically, as an optimization problem with an explicit energy to minimize. Since it deals with multimodal probability distributions until the final step, this approach makes better use of the information that can be extracted from images by machine learning tools. The fact that the problem is solved directly at the global level, with the help of graph cuts, makes the framework more robust to noise and local prediction errors. It also makes it possible to resolve large scale ambiguities which previous approaches could not. This multi-modality framework is not specific to image colorization and could be re-used in any prediction task on images. We used a relatively similar approach for medical imaging purposes, in order to predict computational tomography scans for patients whose magnetic resonance scans are known[15].

We also proposed a way to learn and predict where color variations are probable and how important they are, instead of choosing a spatial coherency criterion by hand, and this performs quite well (see figures 1.3 and 1.6 again). It relies on many features and adapts itself to the current training set.

This colorization approach show significant improvements over [5] and [6], whose examples contain only few colors and lack spatial coherency. The process requires less or similar intervention than [7], and can handle more ambiguous cases and more texture noise. The automatic colorization results should not be compared to those obtained by user-helped approaches since such decisive information is not given here. Nevertheless, since the computational time needed is low, a re-colorization with new color landmarks lasts a fraction of second, and this enables real-time user interaction. This interactive version of the colorization framework clearly competes with the state of the art of colorization with user scribbles[1]. ***(– This supposes to show comparisons with Levin* et al*.)***

## Acknowledgments

We would like to thank Jason Farquhar, Peter Gehler, Matthew Blaschko and Christoph Lampert for very fruitful discussions.

# Bibliography


[1] A. Levin, D. Lischinski, and Y. Weiss, "Colorization using optimization," in *SIGGRAPH '04*, (Los Angeles, California), pp. 689–694, ACM Press, 2004.

[2] L. Yatziv and G. Sapiro, "Fast image and video colorization using chrominance blending," *IEEE Transactions on Image Processing*, vol. 15, no. 5, pp. 1120–1129, 2006.

[3] T. Horiuchi, "Colorization algorithm using probabilistic relaxation," *Image Vision Computing*, vol. 22, no. 3, pp. 197–202, 2004.

[4] T. Takahama, T. Horiuchi, and H. Kotera, "Improvement on colorization accuracy by partitioning algorithm in cielab color space," in *PCM (2)* (K. Aizawa, Y. Nakamura, and S. Satoh, eds.), vol. 3332 of *Lecture Notes in Computer Science*, pp. 794–801, Springer, 2004.

[5] T. Welsh, M. Ashikhmin, and K. Mueller, "Transferring color to greyscale images," in *SIGGRAPH '02: Proc. of the 29th annual conf. on Computer graphics and interactive techniques*, pp. 277–280, ACM Press, 2002.

[6] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin, "Image analogies," in *SIGGRAPH 2001, Computer Graphics Proceedings* (E. Fiume, ed.), pp. 327–340, ACM Press, 2001.

[7] R. Irony, D. Cohen-Or, and D. Lischinski, "Colorization by Example," in *Proceedings of Eurographics Symposium on Rendering 2005 (EGSR'05, June 29–July 1, 2005, Konstanz, Germany)*, pp. 201–210, 2005.

[8] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *9th European Conference on Computer Vision*, (Graz Austria), May 2006.

[9] B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, 2001.

[10] Y. Boykov and V. Kolmogorov, "An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision," in *EMMCVPR '01: Proc. of the Third Intl. Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, (London, UK), pp. 359–374, Springer-Verlag, 2001.

[11] V. Kolmogorov and R. Zabih, "What energy functions can be minimized via graph cuts?," in *ECCV*, pp. 65–81, 2002.

[12] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," in *ICCV*, pp. 377–384, 1999.

[13] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. F. Tappen, and C. Rother, "A comparative study of energy minimization methods for markov random fields," in *ECCV* (A. Leonardis, H. Bischof, and A. Pinz, eds.), vol. 3952 of *Lecture Notes in Computer Science*, pp. 16–29, Springer, 2006.

[14] G. Blasi and D.-R. Recupero, "Fast Colorization of Gray Images," in *Proc. of Eurographics Italian Chapter*, 2003.

[15] M. Hofmann, F. Steinke, V. Scheel, G. Charpiat, J. Farquhar, P. Aschoff, M. Brady, B. Schölkopf, and B. J. Pichler, "MR-based attenuation correction for PET/MR: A novel approach combining pattern recognition and atlas registration," *Journal of Nuclear Medicine*, 11 2008.